# Segmentation and Learning in the Quantitative Analysis of Microscopy Images

Christy Ruggiero[a], Amy Ross[b], Reid Porter[c*]
[a]Nuclear Engineering and Nonproliferation, [b]Nuclear Materials Science,
[c]Intelligence and Space Research
Los Alamos National Laboratory, Los Alamos, New Mexico, USA 87545

## ABSTRACT

In material science and bio-medical domains the quantity and quality of microscopy images is rapidly increasing and there is a great need to automatically detect, delineate and quantify particles, grains, cells, neurons and other functional "objects" within these images. These are challenging problems for image processing because of the variability in object appearance that inevitably arises in real world image acquisition and analysis. One of the most promising (and practical) ways to address these challenges is interactive image segmentation. These algorithms are designed to incorporate input from a human operator to tailor the segmentation method to the image at hand. Interactive image segmentation is now a key tool in a wide range of applications in microscopy and elsewhere. Historically, interactive image segmentation algorithms have tailored segmentation on an image-by-image basis, and information derived from operator input is not transferred between images. But recently there has been increasing interest to use machine learning in segmentation to provide interactive tools that accumulate and learn from the operator input over longer periods of time. These new learning algorithms reduce the need for operator input over time, and can potentially provide a more dynamic balance between customization and automation for different applications. This paper reviews the state of the art in this area, provides a unified view of these algorithms, and compares the segmentation performance of various design choices.

**Keywords:** segmentation, supervised segmentation, image quantification, microscopy, material science

## 1. INTRODUCTION

Fundamental to almost all microscopy analysis, application experts are attempting to compare sets of images generated from materials, be they biological or inorganic, which result from different experimental inputs or collection sources. They increasingly need to rigorously and consistently measure and quantify objects in the images to be able to robustly characterize the features of interest and often need to provide quantitative measurements of the differences between objects in different images in some measurable attribute. Automated or semi-automated image segmentation is critical for the robust and consistent quantification of these attributes.

In material science the quantification need is driven by the demand for more advanced materials, which requires that we can understand and predict material macroscopic properties based on their experimentally driven microstructural and morphological characteristics. In images used for forensics evidence, one of the drivers for this quantitation is the Daubert standard, which requires an understanding of the error rate or uncertainty in any scientific evidence assessment [1]. Accurate segmentation of materials images introduces additional challenges to those encountered in other domains (e.g. bio-medical) and has been the topic of a recent special session [2] and recent references [3, 4]. Some of these challenges include:

1. Material complexity: There are often multiple object types and multiple features (i.e. a variety of structures and sub-structures) in a single image that may be objects of interest. This means the image must be segmented differently depending on the application, and on the user. Images may also lack any background, so techniques that focus on foreground-background segmentation often perform poorly.
2. Material variation: In materials there is often a larger variation in image features within a single object. For example, there may be multiple staining/etching patterns which leads to a large variation in texture, shape, and

* rporter@lanl.gov

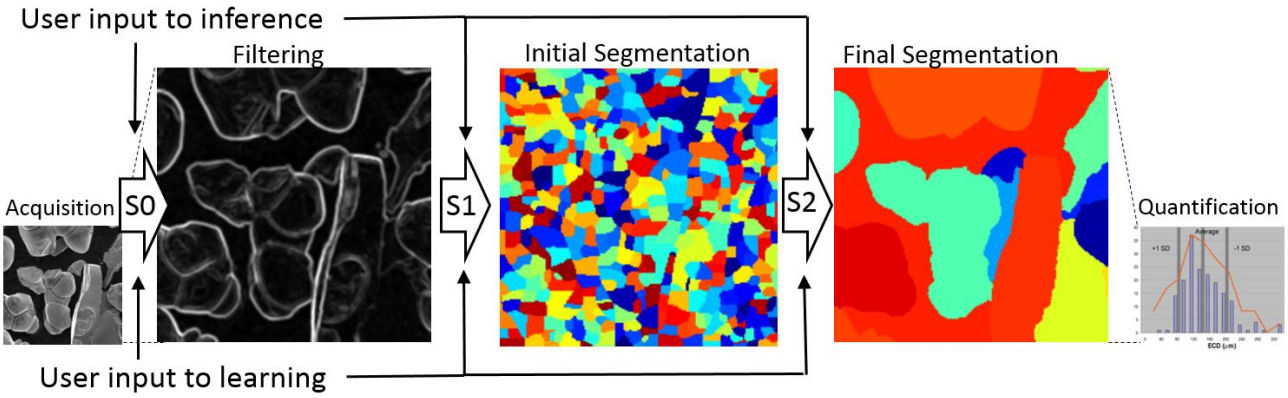


Fig. 1. User input to image analysis workflows in microscopy is typically used in some combination of inference and learning over three distinct stages.

size within a material. This is unlike many biological image analysis problems, where distinct patterns or textures (such as in mitochondria [5]) can be used to assist in segmentation.

3. Material variety: There is a large variety of materials and samples that any one microscopist may want to characterize and compare, and more variation in sample preparation methods than for biological materials. For example, experimental variations may give individuals particulates or agglomerates, which would require different microscopy mounting techniques.

4. Experimental goals: Material scientists often want to vary the materials microstructural and morphological properties as part of the experiment. For biological samples, the experimental goal is seldom to purposely vary the structure and appearance of the biological substances (although, of course, changes in the biological objects do occur and are of course relevant).

5. Image effects: Material microscopy images are notoriously deceiving to human perception. Complex three dimensional objects are imaged at a variety of scales, often with non-homogenous illumination. Pixel values depend on the physical (shape, texture) and chemical (elemental composition and object heterogeneity) aspects of the material as well as the fundamental instrument physics such as detector position and settings (see [6] for many good examples). This again adds a challenge beyond biological substances, which are typically carbon based and aqueous prepared.

In our experience, many materials microscopists still rely on hand (contour-drawing) segmentation, and when software tools are used, these tools often use simple thresholds. Fortunately, with the rapid increase in microscopy imagery in materials applications this is starting to change [4], and much like bio-medical imagery, material science is quickly becoming a significant driver for advanced segmentation tool development. In Section 2 we provide a general overview of how user input is being incorporated into segmentation tools. In Section 3 we describe how a large number of segmentation algorithms can be understood as inference in a graphical model, before describing how machine learning is applied to this framework in Section 4. In Section 5 we present the two stage segmentation architecture that we use to evaluate various design choices. The segmentation performance of these choices on real world material problems is reported in Section 6, before we conclude in Section 7.

## 2. INTERACTIVE SEGMENTATION TOOLS

Segmentation continues to be extensively studied in image and video processing. We focus here on interactive segmentation tools, and how user input is used to improve performance. To facilitate this discussion we will use the analysis workflow illustrated in Figure 1. This workflow breaks segmentation tools into three stages (S0, S1 and S2), and indicates that user input can be utilized in any of these stages in two main ways (inference and learning). Segmentation tools in practice typically use a combination of these components in different ways to solve specific analysis challenges.

**S0: Pre-processing and classification**

Most segmentation algorithms do not work directly with the original image. For example, many algorithms are designed to work on the gradient of the image. In other cases some level of image enhancement is applied. This stage is a good candidate for fine tuning, since image quality and noise characteristics change from one application to the next and even one image to the next. Perhaps the most common way to incorporate user input at this stage is learning. This can be an offline process, where background models are learnt from large corpuses of training data, or online, in where this training data is collected on the fly from users with annotation tools. The most common approach is to train a pixel based classifier, and the most common application is to differentiate foreground from background [7]. In some case the predicted labels can be turned into segments, but these algorithms typically do not attempt to differentiate overlapping or adjacent objects within the image.

**S1: Segmentation**

Stage 1 is where segmentation begins. The most common way to incorporate use input is through inference, and there are two main categories of input. The first input is region centric, and requires users to provide seeds, markers, or scribbles within the objects of interest [8]. The second input is edge centric and typically requires rough delineation, or tracing of object boundaries [9, 10]. Both of these approaches can be very effective for materials images. Learning this stage is less common than stage 0, partly because it is a hard problem. Predicting labels for pixels independently (as is typically done in stage 0, with notable exceptions [11]) is relatively understood, but predicting segmentations is hard and still an active topic of research. We discuss some of these research topics in Section 4. In other directions, learning is beginning to appear in combination with inference in interesting ways. For example, in [12] the user provides 'scribble' mark up of a foreground object, and then active learning guides user to the region in which to scribble next.

**S2: Merging and splitting**

A relatively recent trend is to implement segmentation in two stages. The first stage (our S1) typically uses a fixed method to over segment an image to produce super-pixels. The second stage (our S2) then merges these segments to produce the final segmentation. This approach is a good match for interactive segmentation tools for a number of reasons:

1. The data volume is reduced in the first stage and therefore the second stage is faster and more responsive.
2. Users can interact with segments from the first stage instead of pixels, and this can be a more efficient and intuitive way for users to provide input e.g. merging and splitting of segments [13, 14].
3. The initial segmentation can also be used to generate geometric features for subsequent processing.

Much like stage 1, user input can be used in inference in stage 2, except this time markers constrain the merging of catchment basins [15]. A common approach to exploit user input with learning is to treat potential merges as a tree and then use learning to identify a cut of the tree [16]. However, as we describe in Section 4, this is still a difficult learning problem, and classifiers are typically trained to optimize surrogates for segmentation performance. In other cases researchers have turned to more sophisticated learning frameworks, such as reinforcement learning, so they can directly optimize segmentation performance [17]. As in stage 1, there are also interesting efforts that use active learning in combination with inference to help propagate and reduce the corrections and labels [18].

## 3. SEGMENTATION AS INFERENCE

A very useful way to define segmentation is energy minimization on a graph $G = (\mathcal{V}, \mathcal{E})$. The vertices $\mathcal{V} = \{v_1, v_2, \dots, v_P\}$ are associated with spatial regions (or locations) and edges $\mathcal{E} = \{\dots, e_{ij}, e_{jk}, e_{kl}, e_{lm}, \dots\}$ associate two vertices. In the first stage vertices are associated with pixel locations and the structure of the graph is fixed in advance as a regular grid with edges connecting each vertex to its 4 (or 8) closest neighbors. In the second stage vertices are associated with segments, or super-pixels, and the structure of the graph is dynamic, and irregular, and based on the connectivity of first stage segments. For an image, $X$, the segmentation is defined by labels, $Y = \{y_1, y_2, \dots, y_P\}$, $y_i \in \{1, 2, \dots, K\}$. associated with the vertices. This label identifies the segment to which each location is assigned. With these definitions, we define segmentation algorithms as solving the following inference (or optimization) problem: $\hat{Y} = argmin_Y\, E(Y, X)$. Several popular segmentation algorithms can be expressed as a pairwise energy function:

$$E(Y, X) = \sum_{e_{ij} \in \mathcal{E}} g(y_i, y_j, X). \tag{1}$$

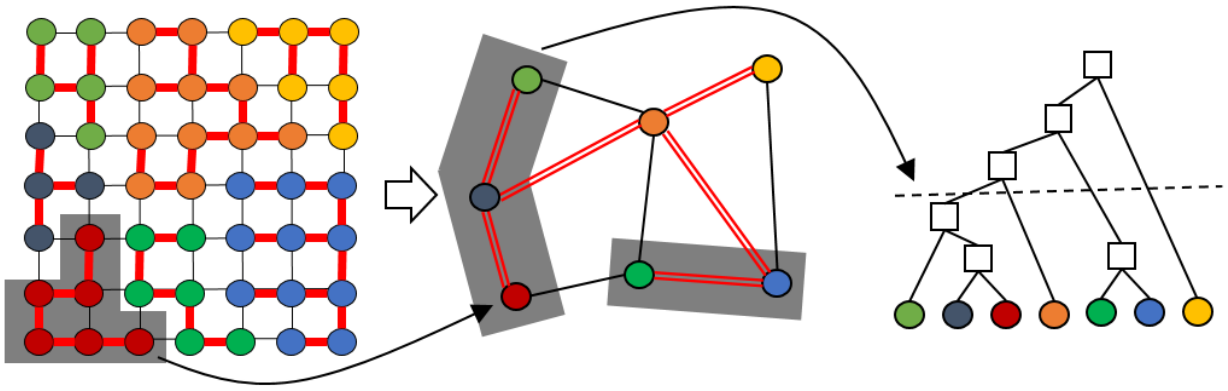

Fig. 2. A popular two stage segmentation method minimizes the Watershed energy function in stage 1 (left). Predicted segments define vertices, and neighboring segments define edges in a second stage graph (middle). It is then common to minimize a Connected Component energy function defined on this graph to produce the final segmentation. Connected Component segmentation can be interpreted as making horizontal cuts of a merge tree (right).

Couprie et. al. show how different algorithms can be obtained with particular choices for $g(\cdot)$ [19] and also show how user input is incorporated into these energy functions. In Figure 1, this corresponds to incorporating user input through inference. One choice of $g(\cdot)$, known as correlation clustering, has the following form [20]:

$$g(y_i, y_j, X) = I(y_i \neq y_j)A(X_{ij}). \tag{2}$$

$I(\cdot)$ is the indicator function that returns 1 when its argument is true, and 0 otherwise. $A(X_{ij}) \in \mathbb{R}$ is a real valued function (often called an affinity function), and $X_{ij}$ is the subset of the image used by the affinity function to determine if pixels $i$ and $j$ should be in the same segment. In principal $X_{ij}$ could be the whole image, but in practice it is typically a local neighborhood that includes $i$ and $j$. $A(X_{ij}) > 0$ indicates pixels $i$ and $j$ should be in the same segment and $A(X_{ij}) < 0$ indicates they should be in different segments. Minimizing Equation 2 is known to be NP hard and therefore there has been much interest in variations (or specializations) of Equation 2 that are tractable. Table 1 provides a summary of some of these efforts. Of particular interest in this paper is watershed segmentation [21], which can be solved efficiently with a minimum spanning forest procedure. Another approach is Connected Component segmentation where the edge weights are restricted to be binary. In this case a minimum can be found by simply discarding the edges with non-positive affinity, and running connected components on the remaining graph. This solution is known as a *perfect clustering* in the correlation clustering literature.

Table1: Specializations of pairwise energy functions that lead to tractable inference and efficient segmentation.

| $g(y_i, y_j, X)$ | Specifics | Inference Method |
|---|---|---|
| $I(y_i \neq y_j)A(X_{ij})$ | $y_i \in \{1, 2, \ldots, K\}$ | Correlation Clustering – NP Hard |
| $I(y_i \neq y_j)A^+(X_{ij})$ | $y_i \in \{0,1\}$ | Graph Cuts – Tractable |
| $I(y_i \neq y_j)I(A(X_{ij}) > 0)$ | $y_i \in \{1, 2, \ldots, K\}$ | Connected Components - Tractable |
| $I(y_i \neq y_j)\left(A^+(X_{ij})\right)^\infty$ | $y_i \in \{1, 2, \ldots, K\}$ | Watershed – Tractable |

Having defined different segmentation algorithms, we return to the two stages of segmentation that were introduced in Figure 1 as S1 and S2. Figure 2 illustrates these two stages in more detail. Mathematically, the architecture can be interpreted as a two layer, recursively defined energy function, similar to Graph-Shifts [22] and other deep networks. From

a practical perspective, this two layer energy function is typically minimized in a feed-forward greedy fashion. One of the most successful and widely used approaches minimizes the Watershed energy in stage 1 followed by Connected Components in stage 2. Another method called the Waterfall algorithm [23], uses the Watershed energy in both (and in fact often multiple) stages. We investigate some of these choices in our experiments.

## 4. SUPERVISED SEGMENTATION

In the first stage, a common choice for the affinity function is the image gradient. In the second stage, a common choice is the depth, area or volume of the first stage segments. However, these are general choices and often lead to segmentations that aren't quite right for particular applications. The solution is to learn the affinity function from training data. The algorithms we describe in this paper are equally applicable to interactive and non-interactive settings. In interactive settings the training data is collected on-line as the user interacts with the segmentation. In non-interactive settings the training data is generated off-line (as in bench-mark datasets). Supervised segmentation requires a loss function to measure how well predictions match with the ground truth. A common choice is based on the Rand Index, which measures how close the predicted labels, $\hat{Y}$, are to the ground-truth labels, $Y$, by counting the number of pairwise differences:

$$\Delta(\hat{Y}, Y) = \binom{P}{2}^{-1} \sum_{i,j \in \binom{P}{2}} (\hat{y}_i \neq \hat{y}_j) I(y_i = y_j) + I(\hat{y}_i = \hat{y}_j) I(y_i \neq y_j). \tag{3}$$

The Rand loss function considers every possible pair of pixels between two images of size $P$ pixels. This can be very expensive to calculate for large images, but for Connected Component segmentation, it can be calculated relatively efficiently. First, we write the edge weight as an edge prediction $\hat{y}_{ij} = I(A(X_{ij}) > 0)$. Then the edge prediction $\{\dots, \hat{y}_{ij}, \hat{y}_{kl}, \hat{y}_{mn}, \dots\}$ is mapped to a vertex prediction $\{\dots, \hat{y}_i, \hat{y}_j, \hat{y}_k, \dots\}$ through a maximin procedure [24]. In words, if any path (max) between vertices $i$ and $j$ is completely connected (min) then connected components will assign $i$ and $j$ to the same segment ($\hat{y}_i = \hat{y}_j$). If $\mathcal{P}_{ij}$ is the set of all paths in $G$ between vertices $i$ and $j$, then Connected Components guarantees:

$$I(\hat{y}_i = \hat{y}_j) = \max_{p \in \mathcal{P}_{ij}} \min_{e_{kl} \in p} \{\hat{y}_{kl}\}. \tag{4}$$

The key to efficient implementation is to realize that for any pair $i, j$ the maximin edge is guaranteed to lie on the Maximum Spanning Tree (MST). Instead of finding each maximin edge $\binom{P}{2}$ times, we can calculate how many times each MST edge appears in the sum, which can be calculated with a modification of Kruskal's minimum spanning tree algorithm. In recent work we presented pseudo-code based on the disjoint set data-structures [25]. Initially, each edge is assigned to its own connected component. A running sum counts the number of times each MST edge prediction should be positive and how many times it should be negative with respect to Equation 3.

The maximin procedure is an extremely useful, and efficient tool for calculating the segmentation error, but it is only applicable to segmentation by Connected Components. This may not be a limitation, since Connected Components is a common choice for the second stage merging. However, in recent work, we showed that the watershed segmentation, can also be cast in terms of Connected Component segmentation, and this is a common choice for the first stage of segmentation [25]. In words, edges that are part of the watershed cut have at least one neighbor on both sides with smaller affinity. More formally, an edge $e_{ij}$ connects two sets of neighboring edges, which we denote $N_i$ and $N_j$. We define $N_{i \setminus j}$ as the set of edges connected to $i$ that excludes edge $e_{ij}$. We use the notation $A_{ij} = A(X_{ij})$ to represent the value of the affinity function at edge $e_{ij}$. The set of edges that belong to watershed basins (those not part of the watershed cut) can be identified with the indicator $I(A^*_{ij} - A_{ij} > 0)$ where:

$$A^*_{ij} = Max\left(Min_{k \in N_{i \setminus j}} A_{ik},\ Min_{k \in N_{j \setminus i}} A_{kj}\right). \tag{6}$$

Once the graph is partitioned, we can apply Connected Components to obtain vertex labels. The only real difference between this approach and the Connected Component segmentation method is we have replaced the global threshold function (which produced horizontal cuts of the merge tree) with a local threshold that implements flooding (and produces non-horizontal cuts of the merge tree).

# 5. TWO STAGE SEGMENTATION

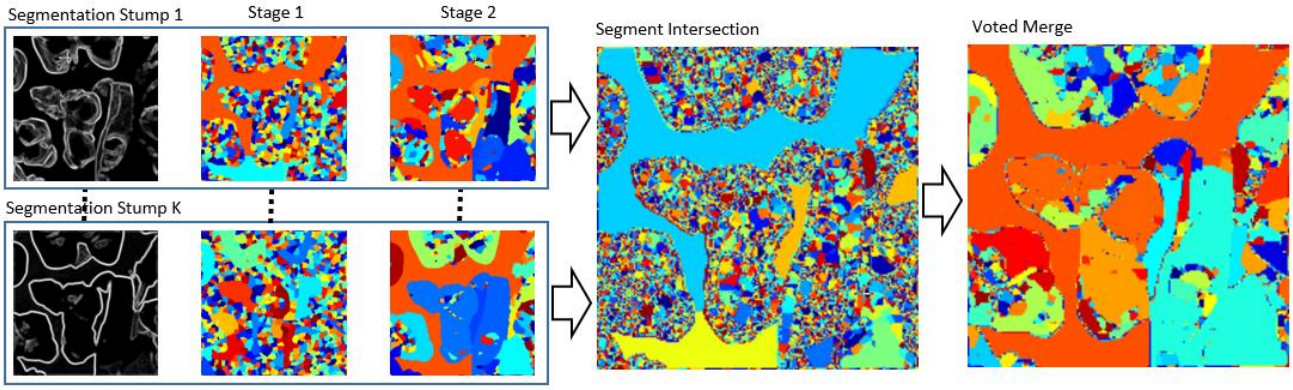


Fig. 3. The segmentation architecture used to compare algorithms is an ensemble of Segmentation Stumps. Each stump implements the two stage segmentation shown in Figure 2. The segmentation predictions are combined through intersection and using a final Connected Components segmentation.

Having a large number of segmentation algorithms in the same framework, and the ability to train affinity functions within these algorithms, opens the door to many interesting questions and potential applications. In this paper we take a first step and try and understand some of the choices that we observe in practice, such as, why does the Watershed, followed by Connected Components do so well and when should we consider alternatives? To help quantify some of these questions we use the segmentation architecture shown in Figure 3, which we now briefly describe.

### 5.1 Segmentation Stumps

The learning method is decision stumps which has no free parameters and is perhaps the simplest learning algorithm possible. We first generate a number of potential features. We generated 12 features in total: 6 features were generated by smoothing the image (with a bilinear filter) at different scales and then estimating the gradient magnitude (with Sobel filters), and 6 features were generated by estimating the gradient magnitude (with Sobel filters) and then smoothing (with Gaussian filters) at different scales. In the first stage the learner enumerates the 12 features, finding a threshold that minimizes Equation 3, and chooses the feature (and threshold) with lowest error. We use one feature for the second stage, and compare two choices. The first is the depth, which calculates the difference between the smallest edge connecting two basins, and the smallest edge within either basin. This is popular choice when using Connected Components in the second stage, partly because it helps the (global) horizontal cut of the merge tree exploit more relative (local) information. When using the Watershed in the second stage, this is less of an issues since the cut is already based on local properties. Instead, a common feature is to use the smallest edge that connects the two basins. Note, that using the same edge feature in both stage 1 and stage 2 is attractive mathematically, since it may simplify recursive definitions of two stage segmentation.

### 5.2 Segmentation Ensemble

Much like decision stumps in standard classification, we expected the Segmentation Stumps to have high bias which could make comparisons between algorithms difficult. We use the same approach used in standard classification to mitigate this effect, which involves treating each stump as a weak learner, and then averaging over an ensemble (bagging). This is not quite as straightforward as it is in standard classification since we need to average over the segmentation (the labeled vertices), not the stump predictions. We use a simple heuristic which seems to work well in practice. First, we find the intersection of all the predicted segmentations. This defines a new segmentation (and graph) where within each segment all the Segmentation Stumps agree. Second, we assign edge weights with the count of Segmentation Stumps that have different labels for the adjacent regions. Since we only want to merge boundaries with low counts, a final segmentation is performed with Connected Components (a horizontal cut of the merge tree), for which we find the best threshold.

# 6. EXPERIMENTS

We apply the architecture in Figure 3 to two different segmentation problems that are encountered in materials images. The Particles problem has multiple overlapping objects on a non-homogenous background. The Grains problem has

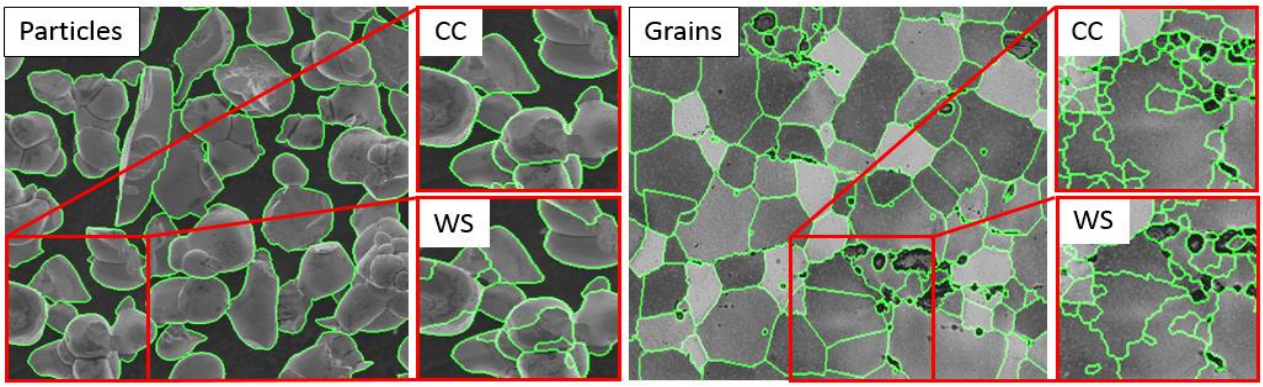


Fig. 4. Two different problems challenge segmentation algorithms in different ways. Ground truth segmentations are illustrated on the larger images and the predicted segmentations in the call-out boxes highlight strengths and weaknesses of using different segmentation algorithms in the second stage of the Segmentation Stump.

multiple, adjacent objects with non-homogenous textures and gradients. Example images and associated ground truth are shown in Figure 4. The ground truth is produced by subject matter experts through manual editing and annotation tools.

We perform a number of experiments to compare various choices within Figure 3. The Watershed (**WS**) segmentation has a default threshold value at 0 (which corresponds to the Minimum Spanning Forrest) and therefore does not need to be trained. When it is trained (**WS-Train**) this threshold can be negative, in which case there are vertices that are not assigned to a catchment basin, or positive, in which case basins separated by low curvature are merged. The Connected Components segmentation however does not have an obvious default value for the threshold, and so it is always trained (**CC-Train**). We compare the various combinations of algorithms and training in the two stages and report results in Table 2. When not specified, we use the **Depth** feature for the Connected Components in the second stage, and the **Min**imum feature for the Watershed in the second stage.

For both Particle and Grain problems there are 4 images. We train using one image and then evaluate on the other 3 images. This is repeated 4 times as we rotate the training image. We average the test scores over all 12 trials to produce the error, and standard error estimates in Table 2. We report the performance of the ensemble, as well as the performance of the best Segmentation Stump.

### 6.1 Discussion

The results in Table 2 indicate the best Stump has lower training error and higher test error compared to the ensemble on both problems, supporting our suspicion that they are highly biased estimators. Also, in almost all problems the voting appeared to improve test set performance, which indicates that although it is currently a heuristic, it may be doing the right thing and worth investigating further. We also observe that training the first stage of the Segmentation Stump was almost always bad for performance. We attribute this to the fact that mistakes in the first stage cannot be corrected by the second stage. We suggest this is particularly problematic when using limited classifiers (such as stumps) in learning and believe this should be reevaluated with more flexible classifiers.

We were at first surprised to see that the best performance on the two problems was obtained with different architectures. The most common two stage architecture (Watershed followed by Connected Components) performed best on the Particles problem, and the Waterfall method (Watershed followed by Watershed) was the best choice for the Grains problem. In Figure 4 we show some example Stumps from these experiments which help explain this result. Depth is a strong cue in the Particles problem and the Connected Component method is able to leverage this to suppress background segmentation with respect to the foreground objects. In the Grains problem the edges are far less defined and the watershed is able to compensate with a second flooding. This is supported by the results using the non-traditional features for each method (last two rows in Table 2) which indicate that depth is a key feature in the Particles problem and not in the Grains problem.

Table2: Summary of experiments with Rand error as percentages and the estimates for standard error.

| Experiment | PARTICLES | | | | GRAINS | | | |
|---|---|---|---|---|---|---|---|---|
| | Ensemble | | Stump | | Ensemble | | Stump | |
| WS, CC-Train TRAIN SCORES | 2.0% | ±0.4 | 1.9% | ±0.4 | 1.5% | ±0.0 | 1.2% | ±0.0 |
| WS, CC-Train TEST SCORES | 2.6% | ±0.5 | 2.7% | ±0.5 | 1.7% | ±0.1 | 2.3% | ±0.4 |
| WS-Train, CC-Train | 4.7% | ±0.9 | 4.3% | ±0.7 | 1.9% | ±0.1 | 2.5% | ±0.5 |
| CC-Train, CC-Train | 3.8% | ±0.7 | 3.7% | ±0.7 | 2.2% | ±0.1 | 2.3% | ±0.2 |
| WS, WS | 5.1% | ±0.5 | 5.5% | ±0.6 | 2.3% | ±0.2 | 1.9% | ±0.1 |
| WS, WS-Train | 3.4% | ±0.4 | 3.6% | ±0.7 | 1.6% | ±0.1 | 2.0% | ±0.2 |
| WS-Train, WS-Train | 3.6% | ±0.4 | 3.6% | ±0.5 | 2.0% | ±0.1 | 2.1% | ±0.1 |
| WS, CC-Train Min | 3.1% | ±0.5 | 3.2% | ±0.7 | 1.7% | ±0.1 | 1.7% | ±0.1 |
| WS, WS-Train Depth | 3.0% | ±2.4 | 2.9% | ±1.7 | 1.6% | ±0.2 | 1.8% | ±0.4 |

## 7. SUMMARY

This paper has shown how a number of segmentation algorithms can be understood as inference and highlighted where learning can be used to incorporate greater user input and potentially produce better segmentations for the challenging microscopy images found in material applications. We have shown that Watershed and Connected Component segmentation algorithms are often used in combination to obtain better results, and our experiments have helped illuminate why this combination often works well. The Watershed provides excellent early stage segmentation by characterizing local, relative variation in the input. Connected Components, with the right features, can then exploit more global information to find the appropriate level in the merge tree. However, we also identified a class of problem in which the Waterfall method appears to be more appropriate. Our future work aims to develop and investigate more general learning algorithms for the Watershed and Connected Component segmentation algorithms. For example, gradient descent could be used to back-propagate errors through both stages of segmentation. This could potentially overcome some of the limitations of greedy feed-forward learning and lead to a more fluid decomposition of segmentation between Watershed and Connected Component stages.

## REFERENCES


1. Saks, M.J. and J.J. Koehler, *The Coming Paradigm Shift in Forensic Identification Science.* Science, 2005. **309**(5736): p. 892-895.
2. Duval, L., et al. *Image Processing for Materials Characterization: Issues, Challenges and Opportunities*. in *ICIP 2014: IEEE International Conference on Image Processing, Special session on Image Processing for Materials Characterization*. 2014. Paris
3. Waggoner, J., et al., *Graph-cut based interactive segmentation of 3D materials-science images.* Machine Vision and Applications, 2014. **25**(6): p. 1615-1629.
4. Beneš, M. and B. Zitová, *Performance evaluation of image segmentation algorithms on microscopic image data.* Journal of microscopy, 2015. **257**(1): p. 65-85.
5. Ghita, O., J. Dietlmeier, and P.F. Whelan, *Automatic Segmentation of Mitochondria in EM Data Using Pairwise Affinity Factorization and Graph-Based Contour Searching.* Image Processing, IEEE Transactions on, 2014. **23**(10): p. 4576-4586.
6. *JEOL Guide to Scanning Microsocope Observation*, JEOL, Editor. 2014. http://www.jeolusa.com/RESOURCES/ElectronOptics/DocumentsDownloads/tabid/320/Default.aspx?EntryId=1.
7. He, J., C.S. Kim, and C.C.J. Kuo, *Interactive Segmentation Techniques: Algorithms and Performance Evaluation*. 2013: Springer Singapore Pte. Limited.
8. Grady, L., *Random walks for image segmentation.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2006. **28**(11): p. 1768-1783.

9. Mortensen, E.N. and W.A. Barrett, *Interactive Segmentation with Intelligent Scissors.* Graphical Models and Image Processing, 1998. **60**(5): p. 349-384.
10. Rother, C., V. Kolmogorov, and A. Blake. *Grabcut: Interactive foreground extraction using iterated graph cuts*. in *ACM Transactions on Graphics (TOG)*. 2004. ACM.
11. Zitnick, P.D.a.C.L. *Structured Forests for Fast Edge Detection*. in *ICCV'13: International Conference on Computer Vision*. 2013.
12. Batra, D., et al., *Interactively co-segmentating topically related images with intelligent scribble guidance.* International journal of computer vision, 2011. **93**(3): p. 273-292.
13. Porter, R.B., S. Lundquist, and C. Ruggiero, *Learning to merge: a new tool for interactive mapping.* Proc. SPIE, 2013: p. 87431F-87431F.
14. Jung, C., et al., *Automatic image segmentation using constraint learning and propagation.* Digital Signal Processing, 2014. **24**(0): p. 106-116.
15. Andrade, M.C. and L.C.M. Pinto, *Interactive characterization of granulated materials.* Computers & Geosciences, 2009. **35**(10): p. 1968-1974.
16. Liu, T., et al. *Watershed merge tree classification for electron microscopy image segmentation*. in *Pattern Recognition (ICPR), 2012 21st International Conference on*. 2012. IEEE.
17. Jain, V., et al. *Learning to agglomerate superpixel hierarchies*. in *Advances in Neural Information Processing Systems*. 2011.
18. Uzunbas, M.G., C. Chen, and D. Metaxas, *Optree: a learning-based adaptive watershed algorithm for neuron segmentation.* Med Image Comput Comput Assist Interv, 2014. **17**(Pt 1): p. 97-105.
19. Couprie, C., et al., *Power Watershed: A Unifying Graph-Based Optimization Framework.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2011. **33**(7): p. 1384-1399.
20. Bansal, N., A. Blum, and S. Chawla, *Correlation Clustering: Theoretical Advances in Data Clustering (Guest Editors: Nina Mishra and Rajeev Motwani).* Machine Learning, 2004. **56**(1-3): p. 89-113.
21. Cousty, J., et al., *Watershed cuts: minimum spanning forests and the drop of water principle.* IEEE Trans Pattern Anal Mach Intell, 2009. **31**(8): p. 1362-74.
22. Corso, J.J., A. Yuille, and Z. Tu. *Graph-Shifts: Natural Image Labeling by Dynamic Hierarchical Computing*. in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 2008.
23. Beucher, S., *Watershed, Hierarchical Segmentation and Waterfall Algorithm*, in *Mathematical Morphology and Its Applications to Image Processing*, J. Serra and P. Soille, Editors. 1994, Springer Netherlands. p. 69-76.
24. Turaga, S.C., et al. *Maximin affinity learning of image segmentation*. in *NIPS*. 2009.
25. Porter, R., D. Oyen, and B.G. Zimmer, *Learning Watershed Cuts Energy Functions*. 2015, Los Alamos National Lab, Technical Report LA-UR-15-20316.